Licenciatura en ciencia de la computación

Algoritmo Euclideano

Matemática Computacional

Profesor:
Nicolas Thériault

Autor:
Sergio Salinas
Danilo Abellá

Introducción

1 Tiempos de ejecución de Algoritmo

A continuación se mostrarán los tiempos de ejecución apra cada valor de "n" y "p" respectivamente.

1.1 Tiempos de ejecución para valores de "n" y "p".

Costos computacionales de los algoritmos donde:

$$p \in \{1/10, 3/10, 1/2, 7/10, 9/10\}$$

para distintos valores de n:

n=18

p=0.1	57.0000000
p=0.3	43.0000000
p=0.5	41.0000000
p=0.7	128.0000000
p=0.9	40.0000000

n=660

p=0.1	18815.0000000
p=0.3	18698.0000000
p=0.5	26974.0000000
p=0.7	18506.0000000
p=0.9	21156.0000000

n=8000

p=0.1	2890310.0000000
p=0.3	3851808.0000000
p=0.5	5042122.0000000
p=0.7	6085456.0000000
p=0.9	6920520.0000000

1.2 Costo computacional para n = 1000.

Costos computacionales de los algoritmos para n = 1000 para distintos valores de p. Tiempos de espera para los siguientes valores de "p":

p=0 34094.0000000
p=0.1 31610.0000000
p=0.2 37190.0000000
p=0.3 55043.0000000
p=0.4 47476.0000000
p=0.5 45258.0000000
p=0.6 46934.0000000
p=0.7 56155.0000000
p=0.8 56874.0000000
p=0.9 47229.0000000
p=1 48239.0000000

2 Formulación experimentos

3 Repretesentación de grafo utilizada

Se utilizo una matriz de adyacencia para representar el grafo debido a que en C se puede manejar sencillamente como un arreglo de dos dimensiones. De esta forma escribir y recorrer la matriz fue tan sencillo como recorrer una matriz de largo n, donde n es la cantidad de vértices que coincide con la cantidad de filas y columnas de va a tener la matriz.

4 Algoritmo de busqueda de camino

La principal razón porque la la búsqueda de ciclo de euler da un ciclo un de euler es debido a que en el grafo generado todas los vértices tienen grado par.

Para crear el algoritmo se utilizo una función recursiva para recorrer el grafo y borrar las aristas entre los vértice y una estrategia de backtracking para volver un paso atras en el caso de que haya llegado a un vértice sin aristas y aun hayan vértice que no se han recorrido. El algoritmo hace lo siguiente.

Al inicio de la función recursiva se posiciona en la fila y columna 0, a partir de ahí comienza a recorrer la fila hasta encontrar un 1, eso significa que hay una arista entre el vértice que esta representando la fila y el vértice que representa la columna, por lo cambia el 1 por un 0, al ser una matriz simétrica se borra en la posición transpeusta también. Después se va a la intercambia la posición de la fila por la columna y se empieza a recorrer esa fila desde la posición 0. el algoritmo continua así hasta que se recorre toda la matriz solo tenga 0 o hasta que se llegue a una fila llena de unos.

En el caso de que llegue a una fila llena de ceros significa que o que se encontró el camino euleriano o que se borraron todas las aristas de ese vertice y aun faltan aristas por recorrer en los otros vertices.

En el primer caso se termina el algoritmo y se muestra el camino. En el segundo se debe hacer un backtracking, para ello se ve el arreglo en el que se guardaba el camino euleriano y se vuelven a poner 1 donde se habían por 0, luego se va a última fila que se recorrio, pero en vez se partir desde la posición 0, esta vez se parte desde la posición siguiente al 0 que se borro, de esta forma se puede seguir recorriendo el grafo sin problemas hasta que ya se haya recorrido todas las aristas.

5 Información de Hardware y Software

5.1 Notebook - Danilo Abellá

5.1.1 Software

- SO: Xubuntu 16.04.1 LTS
- GMP Library
- Mousepad 0.4.0

5.1.2 Hardware

- AMD Turion(tm) X2 Dual-Core Mobile RM-72 2.10GHz
- Memoria (RAM): 4,00 GB(3,75 GB utilizable)
- Adaptador de pantalla: ATI Raedon HD 3200 Graphics

5.2 Notebook - Sergio Salinas

5.2.1 Software

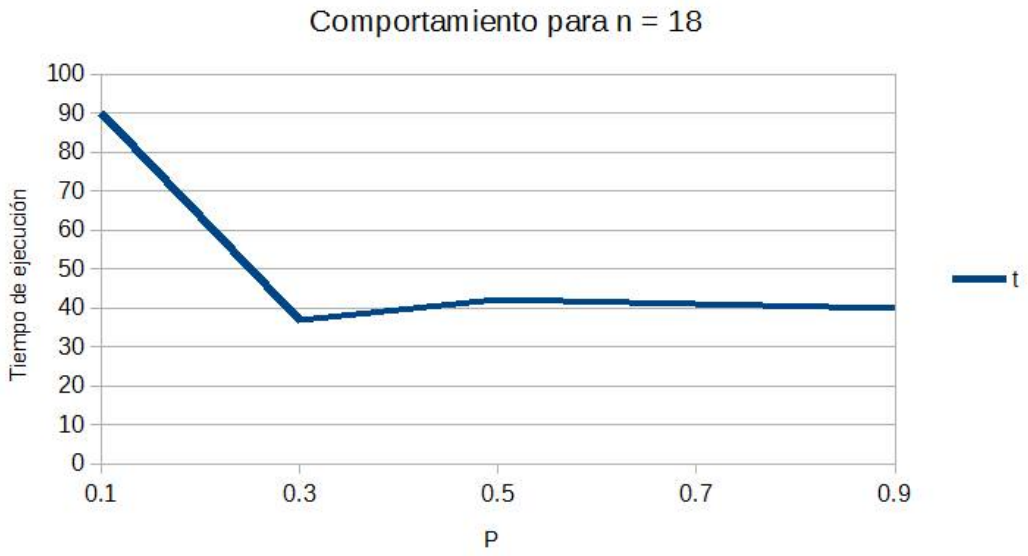
- SO: ubuntu Gnome 16.04 LTS
- Compilador: gcc version 5.4.0 20160609
- Editor de text: Atom

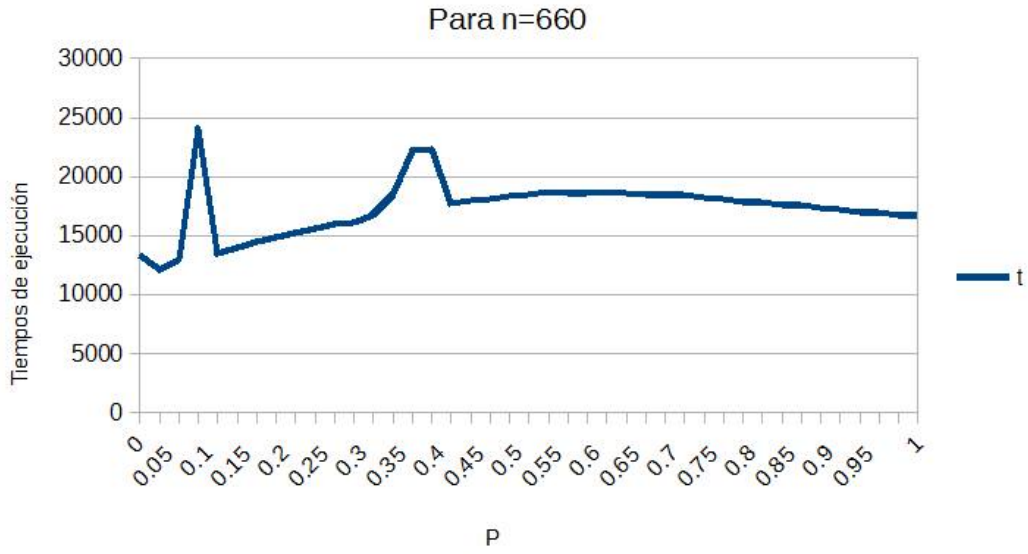
5.2.2 Hardware

- Procesador: Intel Core i7-6500U CPU 2.50GHz x 4
- Video: Intel HD Graphics 520 (Skylake GT2)

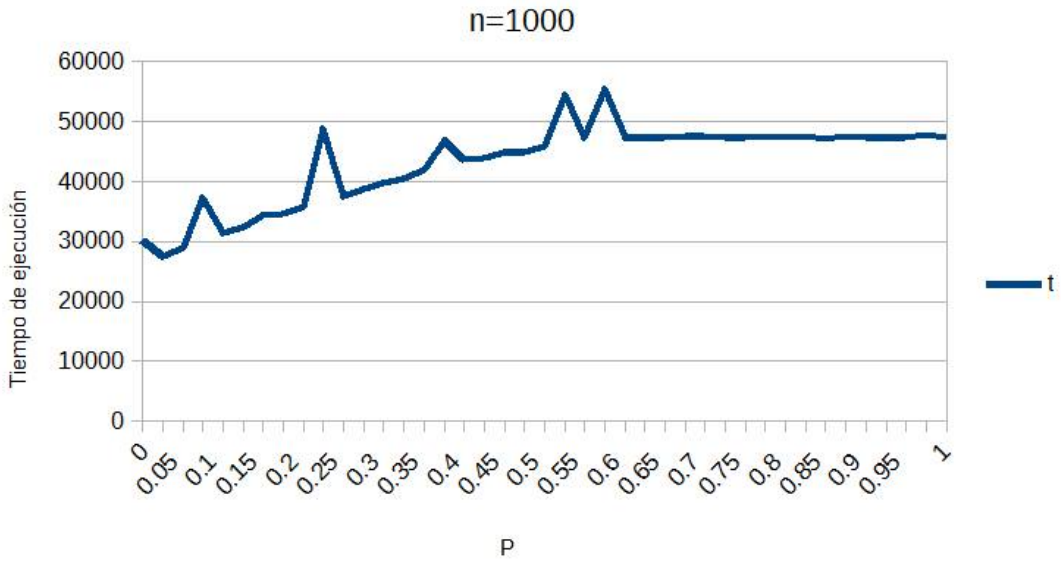
6 Curvas de desempeño de resultados

6.1 Comportamiento para los valores de "p" pedidos.





6.2 Costos computacionales de los algoritmos para n = 1000 para distintos valores de "p".



7 Conclusiones

Con los datos recopilados de los gráficos podemos concluir que mientras mayor sea el valor de ”n” mayor tiempo de ejecución se requiuere.

También cave decir que independiente de ”n” mientras mayor sea el valor de ”p” el tiempo de ejecución también tiende siempre a aumentar , lo que quiere decir que ambas variables influyen en el tiempo de ejecución.

Otro punto a considerar es que cuando ”n” no es tan grande (apróx. menor a 1000) la variable ”p” no es tan influyente , sin embargo cuando ”n” toma un valor grande (apróx. mayor a 1000) ”p” tiene mucha mas influencia en el tiempo de ejecución.